VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

Jakub Šenk

Bakalářská práce

Vedoucí práce: Ing. Marek Běhálek, Ph.D.

Ostrava, 2021

Abstrakt

Tato práce popisuje průběh mé individuální praxe ve firmě KVADOS, a.s. V první části této práce popisuji zaměření a působení společnosti. Dále popisuji mě zadaný úkol, na kterém jsem v rámci této praxe pracoval, a kterým byl návrh a implementace grafické komponenty pro definici výrazů. Závěrem popisuji znalosti, které jsem během praxe získal, celkové zhodnocení přínosu praxe a další potenciál rozvoje dané komponenty.

Klíčová slova

KVADOS, a.s.; Odborná praxe; Angular; TypeScript; .NET; C#; MS SQL; WebAPI; Modulární apliakce

Abstract

This thesis describes course of my professional experience in KVADOS, a.s. company. First part of this thesis describes the company itself and it's focus. Next, I describe my assigned task that I was working on. This task was to design and implement graphical component for expression designing. At last, I describe experience gained during professional experience, overall benefit of the professional experience and future potential of development of this component.

Keywords

KVADOS, a.s.; Professional experience; Angular; TypeScript; .NET; C#; MS SQL; WebAPI; Modular application

Poděkování

Rád bych na tomto místě poděkoval svému nadřízenému Ing. Antonínovi Vaněčkovi za možnost absolvování této praxe a členům týmu, Ing. Vladimírovi Domesovi, Filipovi Šabackému, Ing. Ondřeji Bialasovi, Bc. Martinovi Kováři, Ing. Martinovi Tomisi a Ing. Petrovi Gelnarovi za jejich konzultace, pomoc a cenné rady, protože bez nich by tato práce nevznikla.

Obsah

Seznam použitých zkratek a symbolů

CSS	–	Cascading Style Sheets
CT	–	Content-Type
DOM	–	Document Object Model
HTML	–	HyperText Markup Language
SPA	–	Single-Page Application
UC	–	Use Case

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Již na střední škole jsem začínal pociťovat, že pouze neustálé zkoušení si několika triviálních algoritmů a celkově jen znalost naprostých základů programování mě pro budoucí zaměstnání příliš dobře nepřipraví. Z toho důvodu jsem se rozhodl udělat pro své vzdělání trochu více, a tak jsem nastoupil do čerstvě založené firmy jako stážista. Zde jsem si vyzkoušel programování na zcela jiné úrovni a se zcela jinými technologiemi. Naučil jsem se například s jazykem C#, který mě velice zaujal, a se kterým pracuji dodnes. Dále jsem se zde naučil i technologii Vue.js, což je JavaScriptový framework, který umožnuje velice efektivně vytvářet webové aplikace na velmi pokročilé úrovni.

Jak čas plynul, začaly se projevovat problémy s mým nadřízeným a můj zájem dále pracovat v této firmě postupně opadal. Netrvalo to dlouho a já se ocitl v závěrečném ročníku bakalářského studia a musel jsem si zvolit bakalářskou práci. Byla mi doporučena firma KVADOS, a. s., tak jsem si řekl, proč to nevyzkoušet, jelikož svou dosavadní stáž jsem chtěl ukončit tak jako tak. A tak jsem se rozhodl absolvovat zde bakalářskou praxi jako svou bakalářskou práci.

Zde jsem poznal framework Angular, což je ve své podstatě starší bratr Vue.js, takže naučit se s ním pracovat nebylo zas tolik obtížné. Dále jsem se naučil s TypeScriptem, což je nadstavba JavaScriptu umožňující programovat s předem definovanými datovými typy.

1.1 Popis odborného zaměření firmy a pracovního zařazení

KVADOS, a. s. je společnost zabývající se vývojem softwaru v oblasti obchodu a logistiky. Společnost byla založena roku 1992 s původním názvem HANAS. Za dobu své existence firma publikovala celkem 6 softwarových řešení: VENTUS®, mySTOCK®, myTEAM®, myAVIS®, myCASH® a myDATACENTER®. KVADOS, a. s. nepůsobí jen v české republice, ale také v zahraničí, a to ve více než 11 zemích a k roku 2021 má přes 170 zaměstnanců. Společnost získala několik certifikátů mezinárodních standardů ISO a je také členem IT Clusteru.

Mezi nejvýznamnější klienty této společnosti patří DPO, Marlenka, ČEZ, Démos, Contipro, JIPO Car, Ondrášovka, Henkel, LOREAL, Kofola, Tchibo, R. Jelínek, Česká Pošta, ČD Cargo a mnoho dalších. [1]

Na pohovoru, který proběhl před začátkem školního roku jsem se ucházel o pozici frontend vývojáře. Díky mým předchozím zkušenostem v této oblasti jsem byl přijat do týmu vedeným Ing. Antonínem Vaněčkem, kde jsem se stal desátým členem tohoto malého týmu. Začal jsem se seznamovat s projektem myTEAM® a interními procesy firmy. V rámci odborné praxe mi byl tedy přidělen úkol rozšiřující funkčnost produktu myTEAM®.

1.2 Produkt myTEAM®

myTEAM® byl uveden na trh v roce 2014 a je stále aktivně vyvíjen a zdokonalován. Hlavním cílem tohoto produktu je automatizace a zefektivnění běžných firemních procesů. Jedná se o modulární aplikaci, tzn. že zákazník nemusí mít nutně zaplacenou veškerou funkčnost, ale platí za dílčí celky, které jsou pro něj důležité. V současnosti se myTEAM® skládá z 12 modulů: Podatelna, Porady, Projekty, DMS, Směrnice, Smlouvy, Žádanky, Faktury Přijaté, Nemovitosti, Service Desk, Objednávky Vydané a Vytěžování. Aplikace je postavena na systému panelů, které se skládají za sebe. Neztratí se tak žádné předchozí informace, které jsou tak přehledně uspořádány.

myTEAM® se skládá ze tří hlavních částí z hlediska implementace. Serverové aplikace, klientské aplikace a metadat.

Serverová aplikace, interně nazývána QAS, poskytuje REST API pro klientskou aplikaci. Jedná se o konzolovou aplikaci běžící jako Windows služba.

Klientská aplikace je webová aplikace podporující v současné době již jen prohlížeče Microsoft Edge, Google Chrome a Safari. Ačkoliv aplikace plně funguje i v prohlížeči Mozilla Firefox, tento prohlížeč oficiálně podporován není. Podpora prohlížeče Internet Explorer byla ukončena v průběhu mé praxe.

Metadata slouží jako takový prostředník mezi těmito dvěma aplikacemi. Pomocí metadat se generují některé zdrojové kódy, interně nazývány statickým modelem, a definují se zde parametry pro jednotlivé moduly, jejich panely a nakonec i samotných komponent zobrazovaných v těchto panelech.

1.2.1 Modul DMS

Jelikož má práce byla převážně nad modulem DMS, na úvod jej v rychlosti popíšu.

Tento modul umožňuje efektivní správu a řízení dokumentů, není to však jediné, kromě dokumentů jsou v systému definované Content-Typy (dále jen „CT“) neboli typy obsahu. CT je definice objektu, se kterým uživatel dále běžně pracuje. Aplikace obsahuje předem definované CT jako například Úkol, ale uživatel s příslušnými právy může definovat vlastní CT. CT mají dále své atributy.

Konkrétně u zmíněného Úkolu budeme uvažovat atributy jako například název a popis úkolu, termín splnění, zadavatel, řešitel a spoustu dalších atributů.

Všechny tyto CT (ať už předem definované nebo uživatelsky definované) se mohou zobrazovat v tzv. „Všechnolistu," který se používá pro univerzální fulltextové vyhledávání. Panel, ve kterém se zobrazují výsledky vyhledávání se nazývá Všechnolist. Podle místa a počtu vyfiltrovaného obsahu se může v panelu Všechnolistu zobrazovat více různých CT současně. Pro lepší orientaci může mít každý CT definovaný svůj identifikátor, který se zobrazí ve výsledku vyhledávání pro lepší orientaci. Identifikátorem se rozumí šedý, nevýrazný popisek zobrazený pod názvem daného záznamu. Příklad identifikátoru je vyobrazen na obrázku A.1.

1.3 Použité technologie

Serverová aplikace je napsána v jazyce C#. Jednotlivé moduly jsou realizovány jako DLL knihovny ovládané centrální aplikací běžící jako Windows Služba. Jako úložiště dat je využíván MS SQL Server společně s technologií Elasticsearch.

Klientská aplikace je napsána ve frameworku Angular s podporou TypeScriptu. Jedná se SPA, tzn. že všechny data a prováděné akce jsou zobrazovány a vykonávány v rámci jedné stránky a pouze obsah stránky se mění dynamicky za běhu aplikace. To snižuje přebytečné požadavky na server, takže není nutné posílat po každé akci klientovi zpět celou webovou stránku, jejíž obsah je velice podobný jako obsah předchozí stránky.

Metadata jsou psána ve formátu XML a jsou zpracovávána technologií T4, pomocí které jsou následně generovány výsledné zdrojové kódy. Kromě definice již zmíněných modulů, jejich panelů a použitých komponent se zde dále specifikují například vazby mezi jednotlivými panely, jak si mezi sebou předávají informace a jak se data posílají na server. Každá definovaná komponenta je poté pomocí svého id přiřazena k Angulaří komponentě definované již přímo v HTML souboru daného formuláře na již konkrétním panelu.

1.4 Knihovna Quill

V rámci této práce jsem pracoval i s JavaScriptovou knihovnou Quill [2], kterou na úvod trochu popíšu. Tato knihovna slouží převážně jako richtext editor, tzn. že do vyhrazeného pole je možné psát text, který lze nějak dále upravit a naformátovat. Po označení části textu lze zvolit, zda má tato část textu být tučně, kurzívou, jinou velikosti atd. Do editoru lze také vkládat obrázky, videa a odkazy. Quill nabízí rozsáhlou možnost rozšiřování v podobě modulů a možnosti definovat svůj vlastní formátovací blok.

Quill využívá pro reprezentaci a uložení vstupních dat knihovnu Parchment [3]. Jedná se o stromovou strukturu, která poskytuje dodatečnou funkcionalitu pro Quill editor. Parchment objekt se skládá z tzv. „blotů", kdy každý blot reprezentuje jednotlivý prvek z DOM vykresleném v daném

editoru. Stejně jako se jednotlivé HTML prvky mohou do sebe zanořovat, i bloty se můžou skládat z dalších blotů. Bloty dále mohou poskytovat různé formáty nebo funkčnost. Např. od blotu s názvem Bold bychom očekávali, že obsah daného blotu bude zobrazen tučně.

V blotu může být definovaná jak dodatečná funkčnost, tak stylizace obsahu zobrazovaném v tomto blotu. Jak bude teprve popsáno, skrze blot se například dá definovat zobrazovaný blok, jehož obsah uživatel nemůže editovat, protože může signalizovat místo, kde po dalším zpracování výstupu z editoru bude dosazena dynamicky se měnící hodnota. A tak nemá smysl povolit uživateli přepis obsahu takového bloku, protože vepsaný text by se reálně nikde nezobrazil.

Kapitola 2

Zadání úkolů a jehich časová náročnost

Náplní odborné praxe byl jeden rozsáhlý úkol. Tento úkol pokryl veškerý čas vyhrazený na odbornou praxi. Mým úkolem bylo vytvoření komplexní a podle použitého kontextu konfigurovatelné komponenty na designování výrazů.

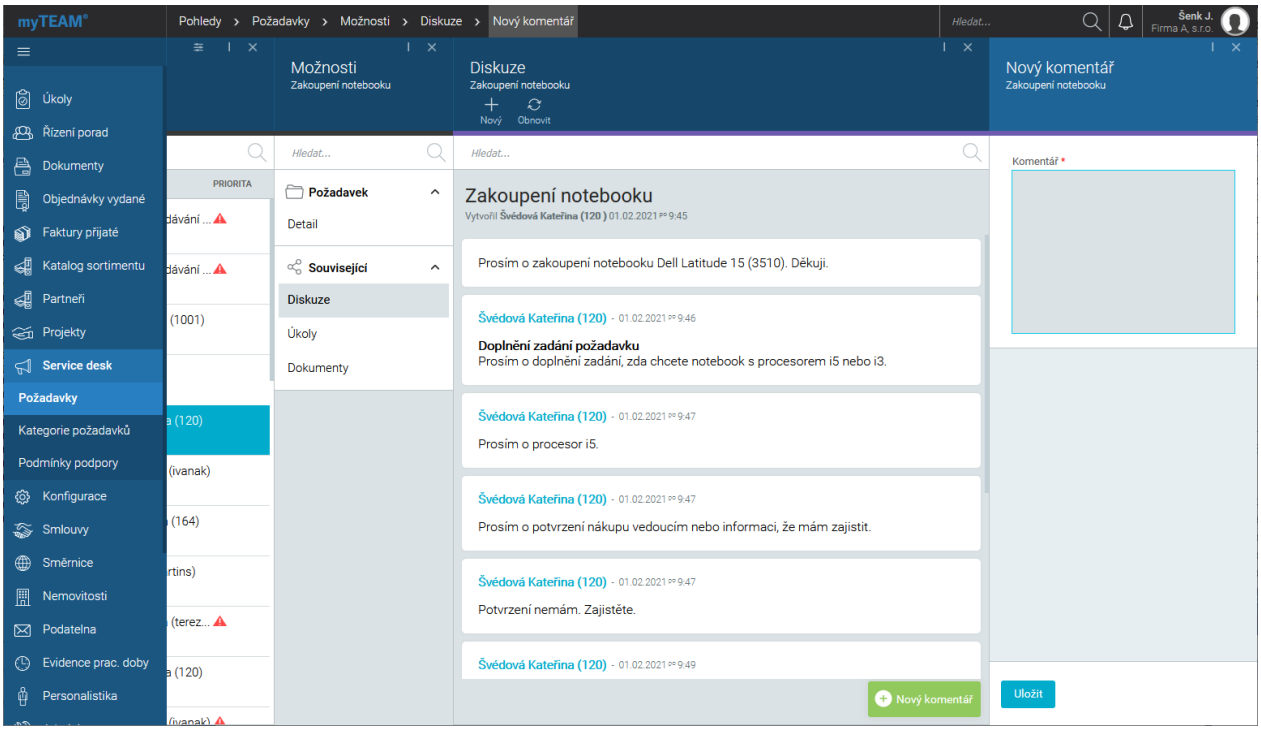
2.1 Prozkoumání dostupných možností

Jelikož požadovaná funkcionalita vyžadovala pokročilý richtext editor, bylo vhodné ověřit, jestli by nebylo možné použít nějakou již existující knihovnu, kterou by bylo podle potřeb možné dále rozšiřovat a konfigurovat. A přesně to byl můj první úkol.

2.2 Ověření funkčnosti

Po zvolení knihovny Quill bylo zapotřebí vyzkoušet, zda bude možné řešení použít v projektu myTEAM®. Bylo nutné provést tzv. „Proof of Concept“. Ten spočíval v zapracování richtext editoru do diskuze.

Modul diskuze se skládá z dvou hlavních panelů. První panel obsahuje seznam komentářů a druhý panel slouží k přidání nového komentáře. Příklad použití diskuze je znázorněn na obrázku 2.1.



Obrázek 2.1: Přehled panelů diskuze

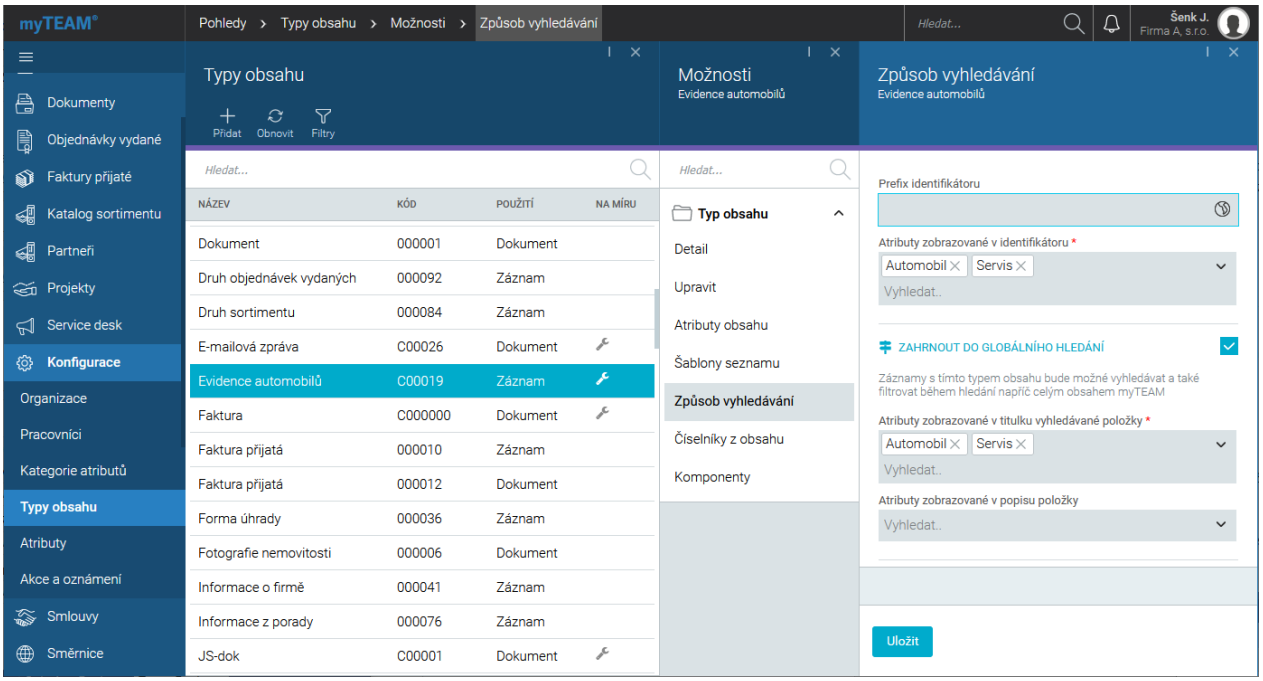
Diskuze je součástí modulu DMS. Což lze pozorovat pomocí barevných proužků pod názvy panelů. Každý modul má svůj vlastní barevný proužek. Lze pozorovat, že aplikace umožňuje napojení různých modulů, které spolu vzájemně komunikují. V tomto konkrétním případě se jedná o modul Servise Desk (černá) a modul DMS (fialová).

Použití editoru v kontextu diskuze by mělo být jednoduché. Editor neměl být nějak výrazně rozšířen o nové funkcionality, jednalo se o pouhou aplikaci pro ověření funkčnosti a integrovatelnosti v projektu myTEAM®.

2.3 Aplikace na identifikátor

Poslední a nejsložitější částí úkolu bylo použití editoru na již zmíněném identifikátoru, který je v aplikaci používán ve Všechnolistu.

Definice identifikátoru byla však poměrně omezená. Uživatel si mohl pomocí dvou polí zvolit prefix identifikátoru a pouhý výčet atributů. S novým richtext editorem by se tyto dvě pole sjednotili do jednoho a uživatel by mohl jednoduše definovat své popisky společně s atributy. Na obrázku 2.2 je vyobrazen panel pro zadávání identifikátoru před úpravou na sjednocený editor.



Obrázek 2.2: Panel pro zadávání identifikátoru

Na obrázku A.1 je poté vyobrazen panel Všechnolistu při výsledku hledání klíčového slova „test“. Lze pozorovat, že výsledky jsou zobrazeny napříč různými CT. Identifikátor, který je zobrazen šedě pod názvem, zde slouží pro dodatečnou informaci o daném záznamu.

Poznámka: Může se zdát, že při vyhledávání identifikátor nepřináší přílišnou dodatečnou hodnotu. Jedná se o testovací data, která nejsou vyplněna na 100% a spousta informací chybí. V produkčních datech, kde je identifikátor řádně vyplněn je orientace ve výsledcích vyhledávání daleko lepší a přehlednější.

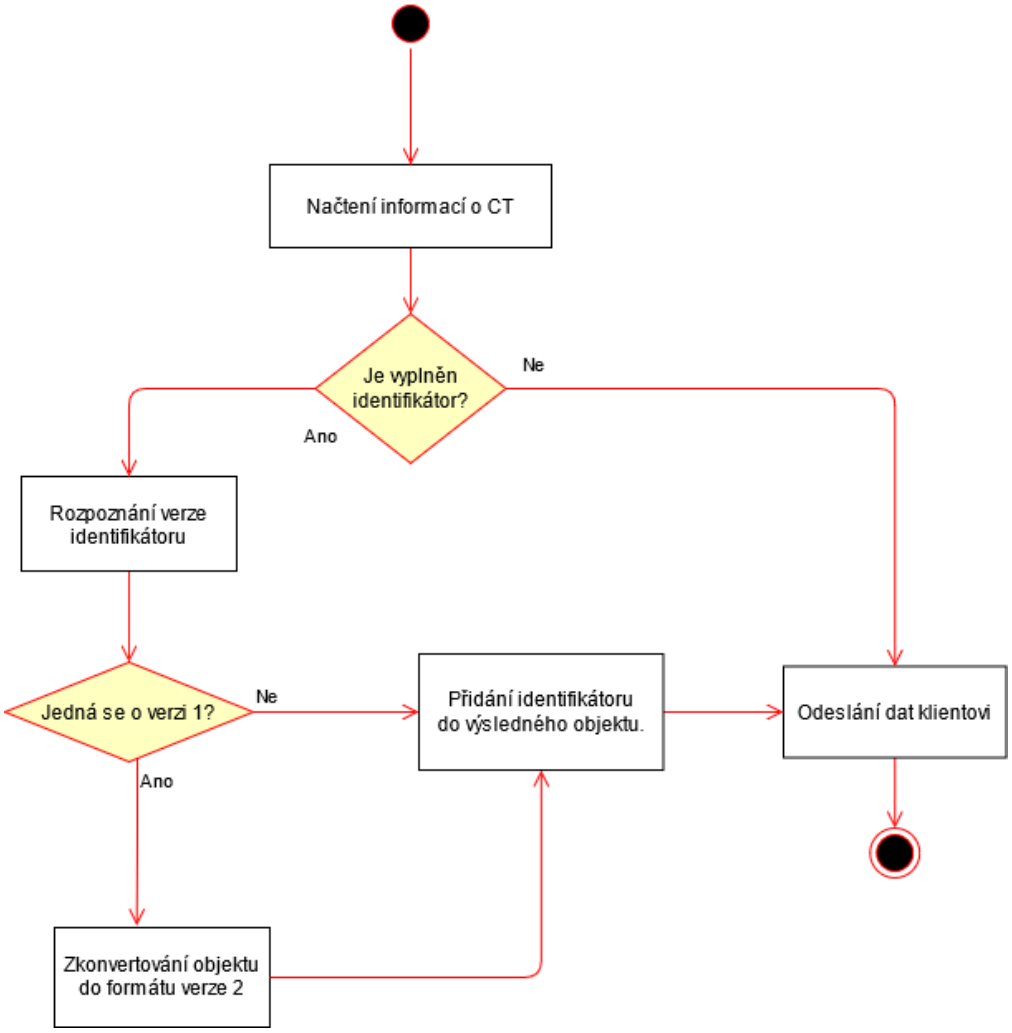
Doba potřebná k přepracování identifikátoru byla odhadnuta na 35 člověkodnů. Přepracování identifikátoru se skládalo z následujících kroků:

Hlavní část práce spočívala ve vytvoření editoru jako takového. Bylo zapotřebí jej ale navrhnout tak, aby jej bylo možné dále a jednoduše znovu použít v různých místech aplikace. Tedy veškeré komponenty editoru musí být psány obecně tak, aby se nezaměřovaly jen na jeden konkrétní případ použití. Také bylo vhodné je navrhnout tak, aby byly jednoduše rozšířitelné v případě dalších potřeb.

Protože se změnil formát ukládání identifikátoru, bylo zapotřebí zajistit zpětnou kompatibilitu tak, aby nový editor dokázal zobrazit a editovat starou verzi definice identifikátoru.

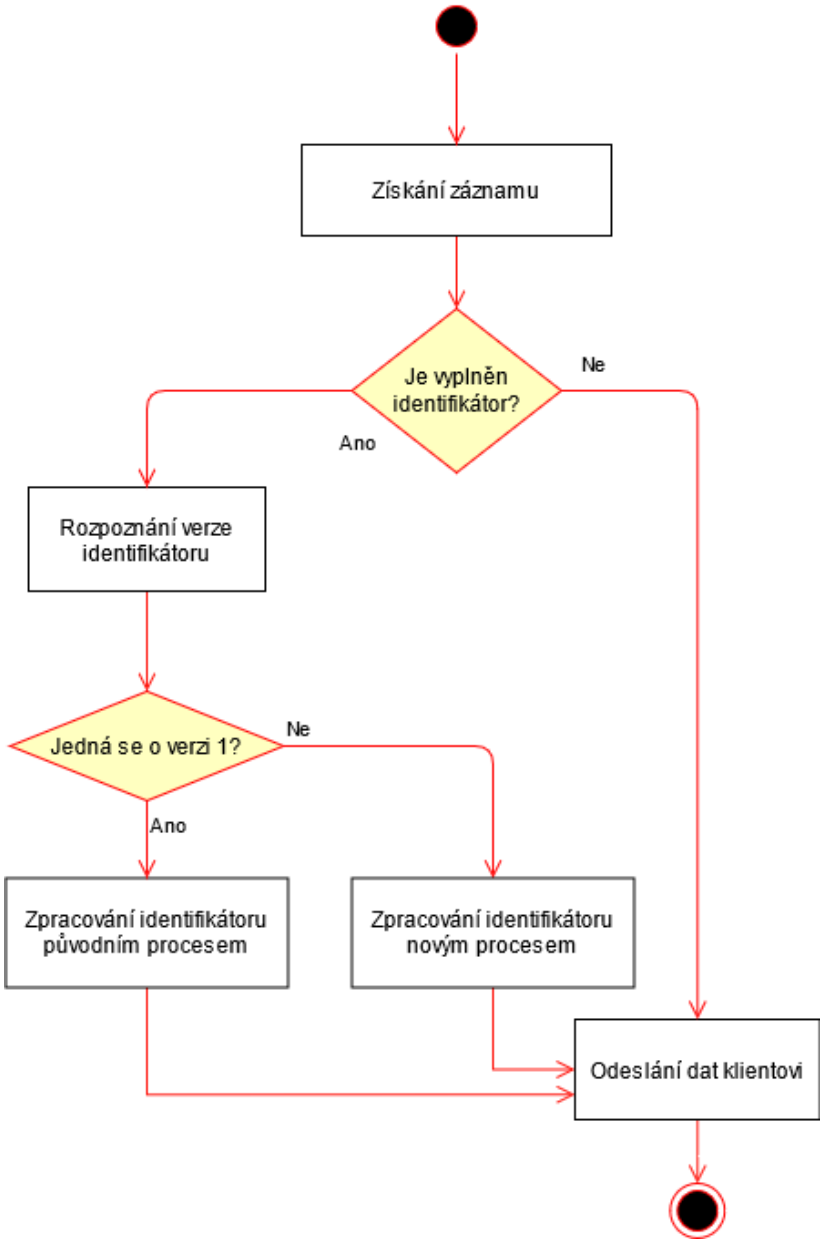
Stejně tak bylo zapotřebí zajistit zpětnou kompatibilitu i při zobrazení identifikátoru v panelu Všechnolistu. Bylo zapotřebí aplikovat správné vyhodnocování a následné zobrazení pro každou verzi zvlášť.

Zjednodušený UC editace identifikátoru je znázorněn na obrázku 2.3.



Obrázek 2.3: UC editace identifikátoru

Na obrázku 2.4 je pak znázorněn zjednodušený UC pro výsledné zpracování a odeslání identifikátoru klientské aplikaci.



Obrázek 2.4: UC zobrazení identifikátoru

2.4 Odhad časové náročnosti

Úkol byl rozdělen na tři samostatné části. První je průzkum možných řešení, druhý je ověření funkčnosti a třetí je aplikace na identifikátor. Odhad časové náročnosti jednotlivých úkolů je znázorněn v tabulce 2.1.

Tabulka 2.1: Časová náročnost zadaných úkolů

Název úkolu	Počet člověkodnů
Průzkum použitelných knihoven	5
Ověření funkčnosti	10
Aplikace na identifikátor	35

Kapitola 3

Řešení zadaných úloh

V následujících sekcích jsou detailněji popsány jednotlivé úkoly a jejich řešení. Poslední části úkolu je věnována samostatná kapitola, jelikož se jedná o nejhlavnější úkol.

3.1 Průzkum dostupných možností

Mým prvním úkolem bylo zjistit, jestli již neexistuje použitelná knihovna, která by splňovala firemní požadavky. Bylo potřeba, aby dostupné řešení umožňovalo v prvé řadě formátování textu jako plnohodnotný richtext editor. Dále bylo vyžadováno, aby editor mohl být jednoduše rozšířen o další funkčnost. Bylo zapotřebí mít možnost definovat vlastní zobrazovací „blok“, který by případně dále reagoval na vstup uživatele a dále komunikoval se zbytkem aplikace.

Jelikož se příliš dobře nedařilo nalézt použitelnou knihovnu, začala se čím dál častěji klást otázka, jestli nebude nutné si napsat vlastní richtext editor zcela od základu. Pokud bych vytvářel vlastní richtext editor úplně od začátku, s velkou pravděpodobností by ve výsledku splňoval naše očekávání, a také by byl napsán na míru naší aplikaci. Fungoval by modulárně, což byl první důležitý požadavek, byl by parametrizovatelný, takže v různých kontextech použití v aplikaci by byly aktivní jiné moduly, a také by reagoval dynamicky na změny. Např. by se načítaly dodatečné moduly podle již dříve vyplňeného pole.

Problém v tomto řešení je samozřejmě cena. Vytvořit takový editor by zabralo velké množství času, pravděpodobně větší, než kolik je vyhrazeno na tuto odbornou praxi.

Spousta nalezených knihoven měla nějaké zásadní omezení, kvůli kterých je nebylo možné použít. Většinu z nakonec nezvolených knihoven omezovala kompatibilita s prohlížečem Internet Explorer. V době, kdy se tento úkol řešil byl ještě prohlížeč Internet Explorer plně podporován, takže z toho důvodu bylo vyžadováno, aby i knihovna tento prohlížeč podporovala. Když už byla objevena knihovna podporující prohlížeč Internet Explorer bylo u ní jiné omezení, a to sice že nepodporovala rozšiřování funkčnosti. Museli jsme si vystačit s pouhým formátováním textu, což nebylo žádoucí.

S další knihovnou, která překonala tyto dvě omezení byl problém v licenci. Potřebovali jsme open source knihovnu, abychom si v případě potřeby mohli samotnou knihovnu upravit.

Nakonec se mi ale podařilo objevit již zmíněnou knihovnu Quill, která od ostatních vyčnívala právě svou modularitou, takže bylo možné tuto knihovnu dále rozšiřovat. Navíc podporovala prohlížeč Internet Explorer, což u velké většiny ostatních knihoven byl problém. Nebylo to ve všech směrech ideální, ale byl to nejlepší kandidát. Rozhodli jsme se tedy pro knihovnu Quill.

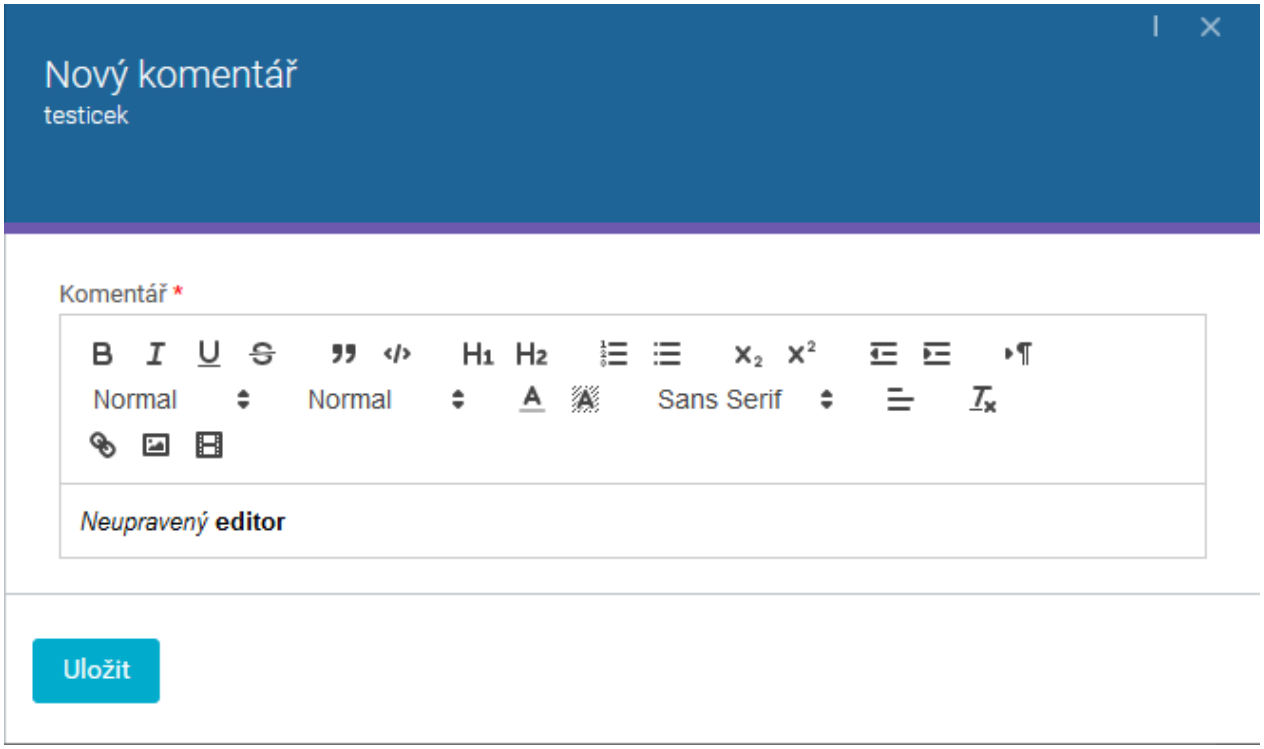
3.2 Ověření funkčnosti

Po odsouhlasení knihovny Quill následovala další část úkolu. Tím bylo předělání modulu diskuze na plnohodnotný richtext editor založený právě na knihovně Quill.

V klientské části aplikace byla využita kromě knihovny Quill také knihovna NGX Quill, která umožňuje lepší integraci a práci s knihovnou Quill ve frameworku Angular.

Jako první byl předělán panel pro přidání komentáře. To proběhlo bez přílišných komplikací. V rámci této fáze byla částečně ověřena rozšiřitelnost a konfigurovatelnost knihovny Quill. Bylo zapotřebí upravit vizuální reprezentaci editoru tak, aby odpovídala standardům zbytku aplikace.

Na obrázku 3.1 je znázorněn Quill editor s výchozí vizuální konfigurací.



Obrázek 3.1: Základní vizualizace Quill editoru

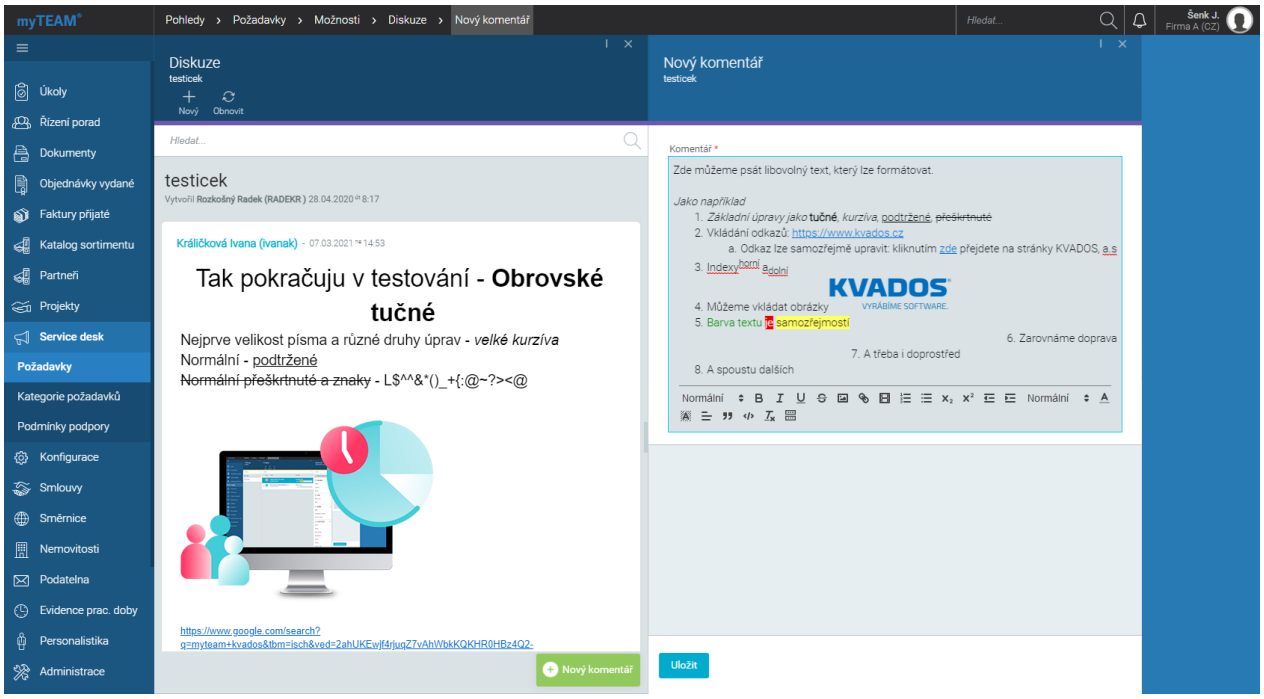
Dále byl předěláván samotný panel s komentáři. Zde jsem zaznamenal první potenciální problém. Problém se zpětnou kompatibilitou. Bylo potřeba zajistit, aby staré komentáře, které byly přidány ještě před úpravou na richtext editor, bylo možné zobrazit v předělaném panelu. Zobrazování richtext komentáře bylo řešeno také skrze Quill, a to tak, že pro každý komentář byla vytvořena nová instance Quill editoru, ovšem s tím rozdílem, že editor fungoval pouze v režimu read-only, tedy nebylo možné komentář již nijak upravovat.

Konečným předěláním panelu se ukázalo, že k žádnému problému se zpětnou kompatibilitou nedošlo. Quill je dostatečně inteligentní na to, aby rozpoznal obyčejný text od svého formátovacího objektu. Takže bez problému a bez dodatečných úprav bylo možné zobrazit jak staré, neformátované komentáře, tak nové, již formátované komentáře v richtextové podobě.

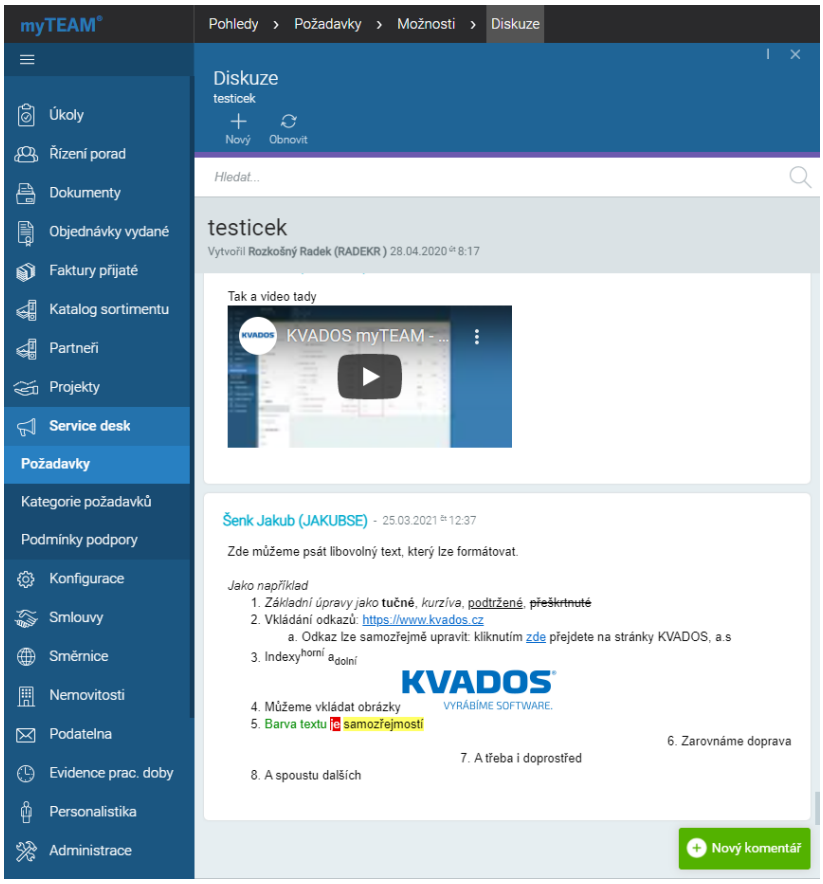
V rámci ověření rozšiřitelnosti knihovny Quill o dodatečnou funkcionalitu byl vytvořen jednoduchý modul, který detekoval vstup uživatele a v případě, že uživatel vloží rozpoznanou url adresu do editoru se daná url adresa automaticky naformátuje do podoby odkazu, takže po uložení komentáře budou moci uživatelé na odkaz kliknout a dostat se tak na požadovanou webovou stránku. Url se také zformátuje, pokud ji uživatel vypíše ručně bez vložení ze schránky.

3.2.1 Výsledek

Na následujících obrázcích 3.2 a 3.3 je znázorněna aplikace po úpravě panelů. Původní editor diskuze je znázorněn na obrázku 2.1.



Obrázek 3.2: Přehled formátovacích možností editoru



Obrázek 3.3: Výsledné zobrazení vstupu

Na obrázku 3.4 je znázorněna celá konfigurace komponenty v případě použití na panelu diskuze.

```
<ControlItem Key="Content" Type="ExpressionDesigner, QASCore/AngularUI">
    <Parameters >
        <Parameter Key="DataSourcePropertyName" Value="StaticModel.K_DMS_COMMENTS.CONTENT_RICH.FullName" ValueMode="StaticModel"/>
        <Parameter Key="Label" ValueMode="Text" Value="DmsCommentEdit.Content"/>
        <Parameter Key="Required" Value="true" ValueMode="Object" ValueType="boolean"/>
        <Parameter Key="Modules" ValueMode="ParameterCollection">
            <Parameters>
                <Parameter Key="AutoLink" ValueMode="TypeRef" Value="AutoLinkDetector, QASCore/AngularUI"/>
            </Parameters>
        </Parameter>
        <Parameter Key="Validator" Value="CompositeValidator, QASCore/AngularCore" ValueMode="TypeRef">
            <Parameters>
                <Parameter Key="RequireValidator" Value="RequiredValidator, QASCore/AngularCore" ValueMode="TypeRef"/>
            </Parameters>
        </Parameter>
    </Parameters>
</ControlItem>
```

Obrázek 3.4: Metadatová konfigurace komponenty

Jak lze pozorovat, konfigurace je poměrně jednoduchá a jednoznačná. V tomto případě jsou zde jen ty nejzákladnější parametry. Byl zde použit jediný modul, který se stará o automatické formátování odkazů. Detailní popis některých parametrů je v tabulce A.2

Kapitola 4

Aplikace na identifikátor

Posledním a nejrozsáhlejším úkolem bylo aplikovat richtext editor na identifikátor CT. Zde opět vznikla otázka zpětné kompatibility. Tentokrát se ale zpětná kompatibilita řešila na straně serveru, nikoliv na straně klienta, tedy Quill si nemohl sám překonvertovávat příchozí objekt do svého formátu. A v tomto případě by to ani nebylo možné, jelikož identifikátor je v databázi uložen zvlášť. Jeden sloupec je vyhrazen pro prefix a v druhém sloupci je serializovaný objektový model popisující výčet použitých atributů. Během návrhu a implementace identifikátoru již bylo bráno v potaz možné budoucí rozšiřování. Z toho důvodu jsou v databázi dodatečné sloupce pro verzi serializovaného objektového modelu a sloupec pro druh serializace.

4.1 Klientská část

Pro zobrazení identifikátoru nebylo nutné používat Quill editor v režimu read-only, protože se identifikátor zobrazuje jako prostý text. Ten ale napřed musí být předem zpracován serverem.

Samotná definice identifikátoru probíhá poměrně jednoduše: do příslušného pole uživatel píše obyčejný text, ten reprezentuje původní prefix, ale je komplexnější, jelikož jej není nutné mít pouze na začátku. V místě, kde má být zobrazen nějaký atribut z CT uživatel může vyvolat dialogové okno pro výběr atributu. Vyvolání dialogového okna je možné provést třemi způsoby.

První je napsání znaku '#' signalizující atribut. Původně se mělo dialogové okno vyvolávat napsáním znaku '@', ale ten je velmi často používán v kontextu označování osob, takže by to mohlo být pro uživatele matoucí. Z toho důvodu byl zvolen znak '#' a znak '@' jsme se rozhodli zachovat pro pozdější implementaci označování osob v modulu diskuze. To už však nebylo součástí této odborné praxe. Druhý způsob vyvolání dialogového okna bylo stisknutím klávesové zkratky Ctrl + Space a třetím bylo obyčejné kliknutí na tlačítko v sekci formátovacích nástrojů.

Zvolený atribut musel být jednoznačně odlišitelný od prostého textu. Bylo možné jej naformátovat, jenže v takovém případě by jej uživatel mohl jednoduše smazat nebo přepsat. Bylo zapotřebí najít způsob, jak uživateli zamezit přepsání vloženého atributu a zároveň uchování dat, které jsou

nezbytné pro pozdější zpracování a rozpoznání daného atributu. I v tomto problému knihovna Quill svou rozšířitelností umožňovala řešení. Je možné definovat vlastní formátovací bloky, tzv. „bloty“. Zde jsem počítal s dodatečným rozšiřováním, a tak jsem vytvořil jeden blot, který je využíván dialogovým modulem pro výběr atributů. Dialogový modul pro výběr atributů je dále popsán v sekci 4.1.1. Druhý vytvořený blot slouží pro reprezentaci samotného formátu a vizualizaci atributu.

Nakonec se však ukázalo, že během vývoje a stálým učením se práce s knihovnou Quill nebylo potřeba dvou samostatných blotů. Při správném rozšíření knihovny byla původní funkčnost řešená druhým blotem převedena a automaticky zpracována samotnou knihovnou její nativní funkcí.

Na identifikátoru situaci komplikoval fakt, že je multijazyčný, tedy pro každý jazyk lze definovat jiný výraz. Při vývoji panelu pro zadávání multijazyčných hodnot nebylo vůbec bráno v potaz budoucí rozšíření a už vůbec ne zadávání nějakých výrazů. Z toho důvodu bylo napřed zapotřebí tento panel upravit tak, aby podporoval původní zadávání textu, ale také zadávání výrazů skrze Quill editor. Upravený panel je sice plně funkční, ale je zde stále velký prostor pro zlepšení, to už ale bude řešeno mimo odbornou praxi.

4.1.1 Dialog výběru atributu

Pro výběr atributu byl vytvořen speciální modul, který rozšiřuje funkcionalitu Quillu. Tvorba tohoto modulu byla poměrně komplikovaná a obtížná, ale ve výsledku to ušetří spoustu času při dalším vývoji a další aplikaci richtext editoru na jiná místa. Dialogový modul byl navržen tak, aby podporoval jakoukoliv formulářovou komponentu.

Jelikož se dialog vytváří dynamicky po vyžádání uživatele, a tedy nemá předem definované místo v HTML souboru, je nutné, aby zobrazovaná komponenta byla taktéž vytvářena dynamicky společně s vytvořením dialogu. Zobrazovaná komponenta v dialogu je tedy definována pouze v metadatech a také nemá své stálé místo v HTML souboru.

Při vytvoření richtext editoru v multijazykovém panelu se zde objevil problém právě s definicí a parsováním metadat, kde je definovaná ona komponenta, kterou má dialog zobrazovat. Jakmile byl richtext editor instanciován v multijazykovém panelu, měl dostupné pouze metadata multijazyčného panelu, a k metadatům panelu s identifikátorem, kde je definovaná komponenta pro dialog přístup neměl. To mělo za následek pád aplikace, protože při vyvolání dialogového okna modul nevěděl, jakou komponentu má vytvořit.

Po dlouhých debatách s ostatními kolegy jsem použil řešení, které přepisovalo kontext dané instance. Při vytváření multijazykového panelu je možné přidat parametry vytvářeným komponentám, a tak byl richtext editorům uměle podsunut kontext panelu s identifikátorem, takže ačkoliv reálně editor existoval v multijazyčném panelu, měl přístup k metadatům nadřazeného panelu, a tedy dialogový modul již byl schopen vytvořit svou komponentu dynamicky.

V případě identifikátoru byla použita komponenta s rozbalovacím seznamem s výčtem atributů přiřazených k danému CT, ale dle parametrů lze dosadit libovolnou komponentu jako například

skupinu zaškrtávacích polí nebo obyčejný textový vstup. Navíc, v případě, kdy by nepostačoval běžný způsob získávání hodnoty z dynamicky zobrazované komponenty, je možné vytvořit vlastní parser, který bude následně využíván dialogovým modulem. Úkolem parseru není nic jiného, než vytažení hodnoty ze zobrazované komponenty a předání ji společně s několika dalšími parametry zpět do dialogového modulu.

Jelikož hodnota rozbalovacího seznamu je objekt obsahující více údajů, nebylo možné použít běžný parser pracující pouze s textem. Vytvořený parser pro práci s rozbalovacím seznamem je vyobrazený na obrázku 4.1.

```
export class AttributeSelectParser implements IParser
{
    public static CreateDataObject(val: any): ExpressionTooltipDataObject
    {
        const o = new ExpressionTooltipDataObject();
        o.DisplayValue = val.Label;
        const data = {};
        data[StaticModel.Constants._Key] = val.Value;
        data[ExpressionTooltip.DataType] = ParserDataTypes.AttributeLookup;
        o.Data = data;
        o.IsEmbed = true;
        o.WithFireChar = false;
        o.Blot = BlockBlot.blotName;
        return o;
    }

    public CreateDataObject(val: any): ExpressionTooltipDataObject
    {
        return AttributeSelectParser.CreateDataObject(val);
    }
}
```

Obrázek 4.1: Parser používaný dialogovým modulem

Parametr val obsahuje výstupní objekt dané formulářové komponenty, pro kterou je navržen. V tomto případě se jedná o rozbalovací seznam, jehož výstupem je objekt obsahující Label jako zobrazovanou hodnotu a Value obsahující id daného atributu. Tyto hodnoty jsou následně přeuloženy do podoby objektu pro dialogový modul společně s dodatečnými parametry. Výsledný objekt je dále zakomponován dialogovým modulem do Quill editoru.

4.1.2 Registrace modulů

Jelikož každý editor bude sloužit k jinému účelu, bylo zapotřebí vyřešit správné nastavení používaných modulů. Všechny moduly se musí napřed zaregistrovat ve knihovně Quill, aby je poté bylo možné používat. Není však žádoucí, aby se registrovaly všechny moduly při prvním načtení panelu s nějakým editorem nebo při načtení samotné aplikace. Do budoucna budou moduly stále přibývat, a tak by načtení stovek modulů, kdy reálně v editoru budou využity třeba jen tři, mohlo negativně ovlivnit výkon celé aplikace. Aby ale rozšiřování bylo jednodušší, byla vytvořena služba pro správu modulů. Každá instance editoru si tak pouze zažádá o určitý modul a služba si poté sama dohledá zbylé informace o daném modulu. V případě, že modul ještě nebyl žádnou instancí editoru vyžádán, služba jej jednoduše zaregistruje.

Původní registrace modulů probíhala následující metodou:

```
Quill.register(<jméno modulu>, <definice modulu>);
```

Registrace skrze nově vytvořenou službu vypadá následovně:

```
this.moduleService.registerModule(<jméno modulu>, <definice modulu>);
```

Použití je velice podobné, ale registrace skrze službu provádí dodatečné kontroly, takže tyto kontroly se nemusí provádět v jednotlivých modulech opakovaně. Aplikace by fungovala i bez této služby a i bez dodatečných kontrol, ale v případě, že by byl zaznamenán pokus zaregistrovat stejný modul znovu nebo zaregistrovat jiný modul pod již použitým názvem, bez dodatečné kontroly by se takto moduly zaregistrovaly a přepsaly by původní modul, což samozřejmě není žádoucí. Opakovaná registrace stejného modulu několikrát by mohla mít negativní vliv na výkon aplikace. A proto se tyto kontroly ve vytvořené službě provádějí.

4.2 Serverová část

Na serverové straně bylo zapotřebí zpracovat příchozí serializovaný objekt. Zde situaci komplikoval fakt, že nebylo jednoduše možné vyřešit zpětnou kompatibilitu tak, aby editor bez problémů zpracoval dříve definované identifikátory starým způsobem.

Ukládání nového identifikátoru bylo poměrně jednoduché. Zde stačilo správně rozlišit různé jazykové varianty, provést případnou validaci vstupu a uložit jej do databáze společně s několika dalšími nastaveními.

4.2.1 Dopad chyby z dávných dob

Komplikace se objevily při zpracování a odeslání identifikátoru uživateli pro editaci. Při načtení objektového modelu z databáze bylo zapotřebí rozpoznat, zda se jedná o verzi 1 (používanou před přepracováním na designér výrazů) nebo verzi 2 (nově používanou designérem výrazů). Na serverové

straně již existoval interface pro práci s verzovanými objekty, který jsem implementoval pro práci s verzí identifikátoru.

Zde se však projevila chyba, která zde byla zavedena při původním návrhu identifikátoru. Stávalo se totiž, že ne vždy měl daný CT s identifikátorem správně vyplněn serializovaný typ a v databázi byla zapsána defaultní hodnota, tedy „unknown“ namísto „json“. Z tohoto důvodu nebylo možné správně rozpoznávat verze identifikátorů a bylo nutné tuto chybu odstranit.

Samotné odstranění této chyby nebylo problém, protože již byla odstraněna přepracováním ukládání identifikátoru. Problém byl v databázi. Předělání ukládání identifikátoru pouze odstranilo chyby, které by vznikly dalším používáním aplikace, ale již uložené identifikátory zůstanou stále špatně založené. Databáze se takto dostala do nekonzistentního stavu. Proto bylo zvoleno řešení tzv. „Startup akcí“. Startup akce slouží k provedení jednorázové operace. Server si pamatuje, které akce již provedl, a které ne. Tudíž při nasazení nové verze u zákazníka server jednoduše rozpozná, že má k dispozici nějaké nové startup akce, které ještě neprovedl, a tak je provede. V nově vytvořené startup akci tedy byl pouze obyčejný UPDATE do databáze, který opravil všechny špatně založené identifikátory.

4.2.2 Verzování

Po opravení této chyby a uvedení databáze do opět konzistentního stavu bylo možné opět začít pokračovat na rozpoznávání jednotlivých verzí.

Bylo zapotřebí podmiňovat, o kterou verzi se jedná a podle toho aplikovat buďto dosavadní nebo novou logiku pro zpracování vstupu a odeslání výsledného textu klientovi.

Podobné podmiňování bylo aplikováno i na místo pro editaci identifikátoru. Jelikož nyní bylo možné editovat identifikátor jen v novém formátu, muselo se opět detekovat, o kterou verzi se jedná. V případě, kdy verze formátu byla 1, tedy ta původní, bylo zapotřebí ji zkonvertovat do podoby verze 2, které klient rozumí a dokáže ji zobrazit uživateli tak, aby ji mohl jednoduše editovat.

Příklady jednotlivých verzí jsou popsány v následujících sekcích.

4.2.3 Původní verze

Původní formát ukládání atributů zobrazených v identifikátoru vypadal následovně:

```
{
  "Items":
  [
    {
      "Attribute": "000799"
    }
  ]
}
```

Je zřejmé, že tento formát neumožňuje přílišnou komplexitu a rozšířitelnost. Vlastní popisek identifikátoru je uložen v databázi v odděleném sloupci a při zpracovávání identifikátoru se výsledné texty pouze spojí a odešlou klientovi.

4.2.4 Nová verze

Příklad formátu verze používané knihovnou Quill vypadá následovně:

```
{
"ops":
  [
    {
      "insert": "SPZ:"
    },
    {
      "insert":
        {
          "BlockBlot":
            {
              "Data":
                {
                  "_Key": "C00846",
                  "dataType": "AttributeLookup"
                },
              "DisplayValue": "SPZ (Evidence automobilů)"
            }
        }
    },
    {
      "insert": " \n"
    }
  ]
}
```

Ops obsahuje pole s veškerými operacemi, které má knihovna Quill provést pro zobrazení takového vstupu. Kromě použité operace insert Quill používá další dvě, a to jsou retain a delete. Je zřejmé, že pokud chceme zobrazit vstup v editoru určený pro úpravu uživatelem, budou se operace skládat z jednotlivých insert operací. Operace retain a delete se používají interně Quill knihovnou již pro samotnou editaci načteného vstupu.

4.2.5 Uložení do databáze

Posledním problémem na serverové straně bylo samotné uložení vstupu do databáze. Jelikož vstup uživatele v sobě obsahuje nejen id a další vlastnosti atributu, ale také nese informaci, v jakém místě se atribut zobrazuje, a také zobrazovaný název. Je ale špatný nápad ukládat do databáze zobrazovaný název atributu jako součást většího celku, protože atribut se může přejmenovat, ale zobrazovaný název by se nezměnil. Databáze by se tak dostala opět do nekonzistentního stavu.

Z toho důvodu byla vytvořena služba umožňující mutaci vstupu. Služba umožňuje transformaci vstupu do databázového formátu a zpět. Transformace probíhá způsobem, kdy služba pouze hledá výskyty datové reprezentace atributu a ke každému výskytu odmaže jeho zobrazovaný název, výsledný vstup se poté uloží do databáze. Transformace zpět probíhá obdobně. Služba hledá datovou reprezentaci atributu. Podle jeho id zjistí jeho aktuální název a na příslušné místo doplní hodnotu s aktuálním názvem. Taktéž obnoví odstraněné formáty a tímto je načtený vstup z databáze připraven na odeslání zpět klientovi.

Po zpracování vstupu z předchozí sekce touto službou by tedy výsledek uložený do databáze vypadal takto:

```
{
"ops":
  [
    {
      "insert": "SPZ:"
    },
    {
      "insert":
        {
          "BlockBlot":
            {
              "Data":
                {
                  "_Key": "C00846",
                  "dataType": "AttributeLookup"
                }
            }
        }
    },
    {
      "insert": " \n"
    }
```

```
  ]
}
```

Kód použitý pro mutaci vstupu je poměrně triviální a je znázorněn na obrázku 4.2.

```
public string PrepareForDB(string quillDelta)
{
  if (string.IsNullOrEmpty(quillDelta)) return quillDelta;
  JObject d = JsonConvert.DeserializeObject<JObject>(quillDelta);
  foreach (JObject item in d.Value<IEnumerable>(ops))
  {
    if (item.GetValue(insert) != null)
    {
      if (item.GetValue(insert).HasValues && item.GetValue(insert).Value<JObject>(BlotTypes.BlockBlot) != null)
      {
        item.Value<JObject>(insert).Value<JObject>(BlotTypes.BlockBlot).Remove(DisplayValue);
      }
    }
    else throw new ArgumentException("Quill delta does not contain insert.");
  }
  return Trim(JsonConvert.SerializeObject(d));
}
```

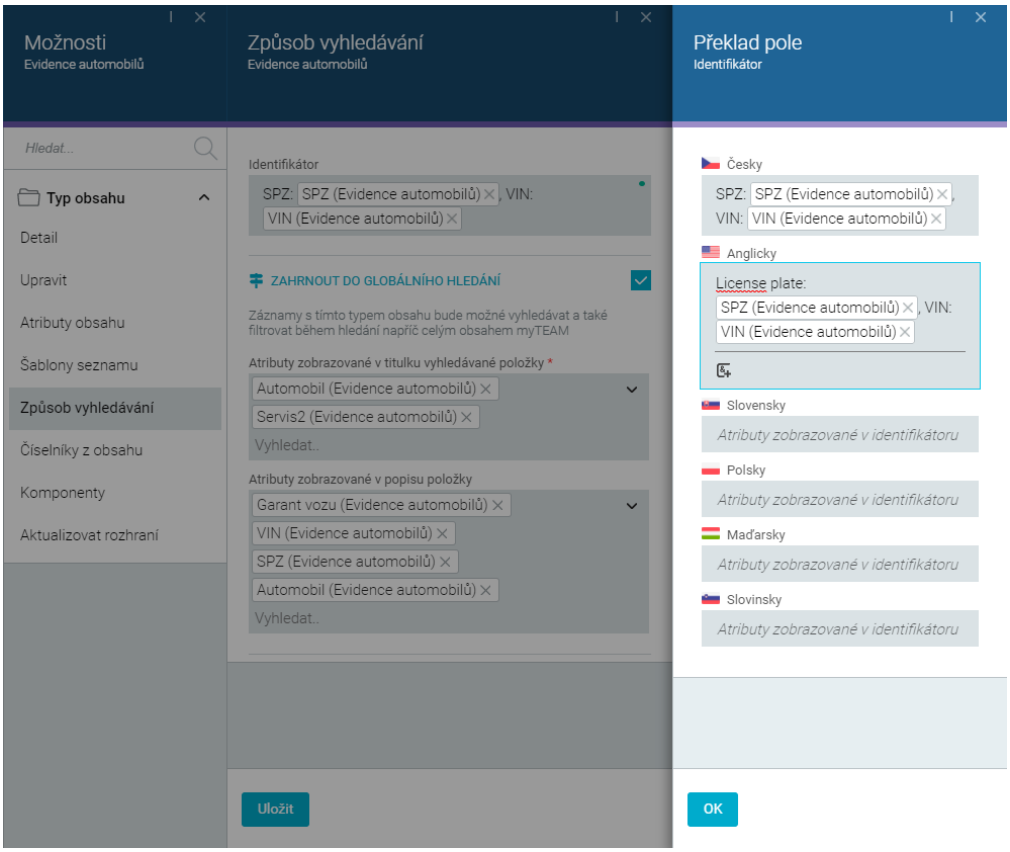
Obrázek 4.2: Metoda provádějící mutaci vstupu

4.2.6 Zobrazení ve Všechnolistu

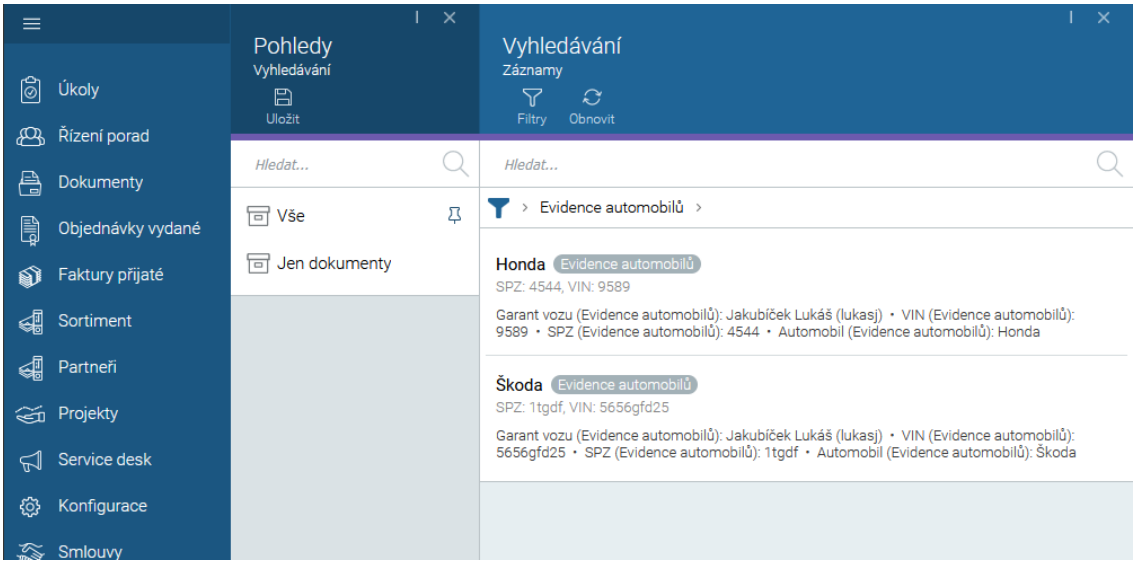
Poslední část úkolu spočívala ve zpracování a zobrazení výsledného textu reprezentující identifikátor. Zpracování probíhalo poměrně jednoduše. Na serverové aplikaci jsou CT ukládány do paměti, takže při vyfiltrování záznamu je k dispozici definice celého CT, a tedy i jeho identifikátoru. Ten je již uložen v podobě verzovaného objektu, a tak je možné provést běžnou kontrolu, jestli je daný identifikátor ve verzi 2 nebo 1.

4.3 Výsledek

Na následujících obrázcích 4.4 a 4.3 je znázorněna aplikace po úpravě identifikátoru.



Obrázek 4.3: Nový způsob definice identifikátoru



Obrázek 4.4: Výsledné zobrazení identifikátoru

Na obrázku 4.5 je znázorněna výsledná konfigurace komponenty použité na panelu s identifikátorem.

```
<ControlItem Key="IdentifierSearchFormatDesigner" Type="ExpressionDesigner, QASCore/AngularUI">
    <Parameters>
        <Parameter Key="Label" ValueMode="Text" Value="DmsContentTypeSearchTemplateEdit.ViewIdentifierTitle"/>
        <Parameter Key="LabelPosition" ValueMode="Object" ValueType="LabelPosition, QASCore/AngularUI" Value="Above" />
        <Parameter Key="Placeholder" ValueMode="Text" Value="DmsContentTypeSearchTemplateEdit.ViewIdentifierPlaceholder"/>
        <Parameter Key="Multilanguage" Value="true" ValueMode="Object" ValueType="boolean"/>
        <Parameter Key="MetadataSourcePropertyName" Value="DmsSearchTemplateReducerDataViewConstants.IdentifierSearchFormatParametersOutputItem, ..." />
        <Parameter Key="DataSourcePropertyName" ValueMode="StaticModel" Value="StaticModel.K_DMS_CONTENT_TYPES.IDENTIFIER_FORMAT.FullName" />
        <Parameter Key="Modules" ValueMode="ParameterCollection">
            <Parameters>
                <Parameter Key="AttributeTooltip" ValueMode="TypeRef" Value="QuillTooltipModule, QASCore/AngularUI">
                    <Parameters>
                        <Parameter Key="ComponentKey" ValueMode="UIComponentRef" Value="\View\Panel\Form\AttributeSelect"/>
                        <Parameter Key="ComponentType" ValueMode="TypeRef" Value="DataViewSelect, QASCore/AngularUI"/>
                        <Parameter Key="ValueParser" ValueMode="TypeRef" Value="AttributeSelectParser, QASCore/AngularUI"/>
                        <Parameter Key="UseBlot" ValueMode="TypeRef" Value="BlockBlot, QASCore/AngularUI/FormComponents/ExpressionDesigner/Blots/BlockBlot" />
                        <Parameter Key="ToolbarIcon" ValueMode="ImageRef" Value="core_form_richtext_attribute.png, Modules.Core.MD" />
                        <Parameter Key="IconTooltip" ValueMode="Text" Value="DmsCommon.Attribute" />
                        <Parameter Key="FireChar" ValueMode="Object" ValueType="string" Value="#" />
                        <!-- 32 is space-->
                        <Parameter Key="FireKey" ValueMode="Object" ValueType="number" Value="32" />
                        <Parameter Key="CtrlKey" ValueMode="Object" ValueType="boolean" Value="true" />
                    </Parameters>
                </Parameter>
            </Parameters>
        </Parameter>
        <Parameter Key="DefaultOptions" Value="false" ValueMode="Object" ValueType="boolean"/>
    </Parameters>
</ControlItem>
```

Obrázek 4.5: Konfigurace komponenty

Zde už je konfigurace o něco delší. Detaily jednotlivých parametrů jsou popsány v tabulce A.2. V tabulce A.1 jsou poté popsány používané datové typy.

Kapitola 5

Zhodnocení praxe

V poslední části této práce je zhodnocen průběh celé praxe, dosažený výsledek, scházející a získané znalosti.

5.1 Znalosti získané během studia a uplatněné v praxi

Ačkoliv náplní práce byla spíše oblast webových aplikací, která se probírá v předmětu vývoj internetových aplikací, tento předmět mi příliš nepomohl, protože základní syntaxi JavaScriptu, HTML a CSS vyučovanou v tomto předmětě jsem již znal. TypeScript byl jen velmi okrajově zmíněn a Angular nebo jiné frameworky se zde neprobíraly vůbec.

Největším přínosem byl pro mě předmět architektura technologie .NET, kde jsem v praxi uplatnil převážně znalost pokročilých možností jazyka C#, a tedy jsem dokázal porozumět některým složitějším funkcím na serverové straně, které jsem při práci využíval.

Dále pro mě byl přínosný předmět vývoj informačních systémů, kde jsem získal základní povědomí o navrhovaných architekturách při vytváření softwarů.

5.2 Scházející znalosti

Neoddělitelnou součástí praxe byla také práce v týmu. Během vývoje bylo zapotřebí se několikrát sejít a prodiskutovat problém, jestli jej řešit způsobem A nebo způsobem B. Zde se ukázalo, že je pro mě poměrně obtížné srozumitelně vysvětlit, o jakou problematiku se jedná, jaké jsou případné možnosti řešení a co by obnášelo takové řešení použít.

Takže určitě potřebuji zlepšit své komunikační dovednosti, které jsou pro práci v týmu velice důležité.

5.3 Závěr

Výslednou komponentu jsem předal svému vedoucímu k další analýze, jelikož během vývoje jsem průběžně zjišťoval, že potenciál a celkové využití této práce jsou mnohem větší, než jsme si na začátku mysleli, a tak po úspěšném ukončení této odborné praxe bylo naplánováno další využití a rozvoj této technologie.

Celkový průběh a přínos celé praxe hodnotím velice pozitivně. Zdokonalil jsem si znalost jazyka C# a webových technologií, konkrétně Angular a TypeScript, a také jsem se naučil pracovat s knihovnou Quill.

Kromě toho jsem získal i základní povědomí o tom, jak lépe navrhovat architekturu softwarových řešení, což je určitě velmi cenná zkušenost.

Literatura

1. *KVADOS, a. s.* [Online] [cit. 2021-04-05]. Dostupné z: `https://kvados.cz/`.
2. *Quill* [online] [cit. 2021-04-05]. Dostupné z: `https://quilljs.com/`.
3. *Parchment* [online] [cit. 2021-04-16]. Dostupné z: `https://github.com/quilljs/parchment`.

Příloha A

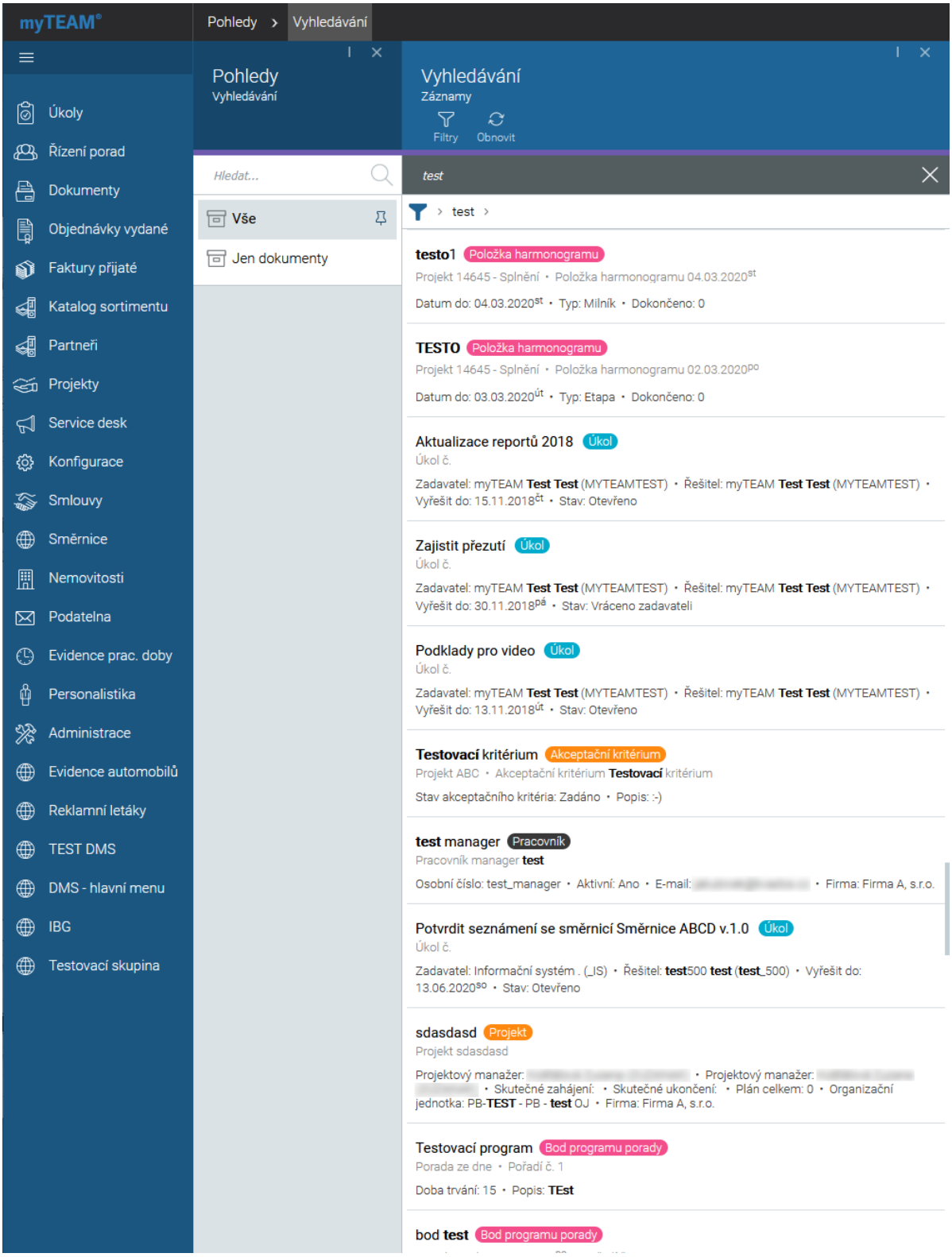
Přehled konfigurace

Tabulka A.1: Popis datových typů

Datový typ	Popis
Text	Reference do tabulky překladů, kde je dosazen překlad podle zvoleného jazyka.
Object/LabelPosition	Enum určující pozici.
Object/boolean	Standardní JS boolean: true nebo false.
StaticModel	Výsledkem je běžný string, použitá konstanta je nahrazena během kompilace pro zabránění překlepů.
ParameterCollection	Dictionary s vícero parametry.
TypeRef	Reference na konkrérní datový typ/třídu.
UIComponentRef	Reference na libovolnou komponentu a její metadata.
ImageRef	Reference na obrázek z daného modulu.
Object/string	Standardní JS string.
Object/number	Standardní JS number.

Tabulka A.2: Popis konfiguračních parametrů

Název parametru	Typ	Popis parametru
Label	Text	Popisek zobrazený u komponenty.
LabelPosition	Object/LabelPosition	Pozice popisku.
Placeholder	Text	Nevýrazný text, který zmizí po vepsání prvního znaku.
Multilanguage	Object/boolean	Urřuje, zda při kliku se otevře panel pro zadávání více jazyků.
MetadataSourcePropertyName	StaticModel	Hodnota používaná reducerem. V rámci ochrany firemního know-how tuto technologii nebudu dále popisovat.
DataSourcePropertyName	StaticModel	Klíč, pod kterým je hodnota komponenty uložena.
Modules	ParameterCollection	Seznam aktivních modulů.
AttributeTooltip	TypeRef	Typ použitého modulu s dalšími parametry.
ComponentKey	UIComponentRef	Reference na komponentu, která má být zobrazena v dialogu.
ComponentType	TypeRef	Typ komponenty zobrazené v dialogu.
ValueParser	TypeRef	Parser, který vytáhne hodnotu zobrazené komponenty.
UseBlot	TypeRef	Formátovací blok, který má být použit po vybrání a vložení hodnoty z dialogu.
ToolbarIcon	ImageRef	Ikona tlačítka v sekci formátovacích nástrojů.
IconTooltip	Text	Popisek zobrazený po najetí na ikonu v sekci formátovacích nástrojů.
FireChar	Object/string	Po napsání specifikovaného znaku se vyvolá dialog.
FireKey	Object/number	Po stisknutí specifikované klávesy se vyvolá dialog.
CtrlKey	Object/boolean	Jestli má být současně s klávesou zmáčknuta klávesa Ctrl.
DefaultOptions	Object/boolean	Jestli se mají zobrazovat standardní formátovací nástroje.



Obrázek A.1: Vyobrazení identifikátoru v panelu Všechnolistu